MPC-684 Programing tutorials

ACCEL
Last update May 24,2005

(MPC-684F)

This tutorial is an outline for MPC-684 programming.
Please see the “MPC-684 User’s manual” for details about products and commands.
For the newest information you should look at the web site http://www.accelmpc.co.jp

Index

MPC-684 family

◆ Outline of MPC-684 series

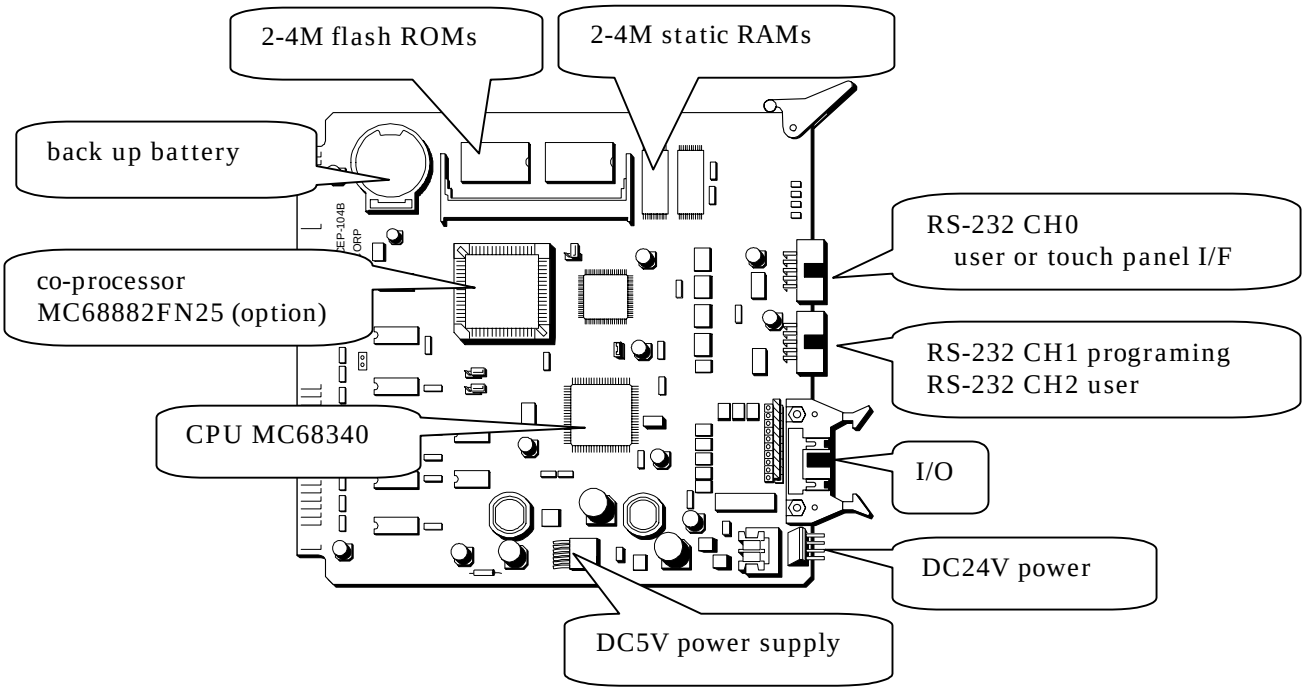
MPC-684	main CPU board	serial communication port (RS-232) 3ch 4 outputs 8 inputs
MOP-096	output board	96 transistor open-collector outputs max 4 boards in a system.
MOP-048	output board	48 transistor open-collector outputs max 8 boards in a system.
MIP-096	input board	96 photo-coupler isolated inputs max 4 boards in a system.
MIP-048	input board	48 photo-coupler isolated inputs max 8 boards in a system.
IOP-048	input/output board	24 photo-coupler isolated inputs, 24 transistor open-collector outputs.
MPG-314	pulse generation board	4-axis. max seed 4Mpps s-curve acceleration / deceleration, max 10 boards
MBK-SH	touch panel interface	Digital corporation's touch panel communication board.
MRS-402	serial communication board	RS-232 or RS-485 2ch. max 2 boards.
MPC-SLNK	remote I/O board	SUNX corporation's S-LINK system support. 512 inputs/outputs. max 2 board.
RACK-68K3	system rack	3 slots
RACK-N6	system rack	6 slots
RACK-N13	system rack	13 slots

For example

If you combine MPC-684, IOP-048, MPG-314 and RAC-K-68K3 you can build 2 user serial communication ports, 48 inputs/outputs, 4 axes pulse controller.

The feature of the MPC-684

- Program capacity 500Kbytes(about 25000 lines)
- Point data 4 by 13000 points
- Global variable 2000
- Local variable 26 (each task)
- Task from 0 to 31
- RS-232 buffer 256 bytes each ports
- Real time clock not supported but you can use the touch panel clock
- I/O 8 inputs 4 outputs
- Loading power supply DC5V 2A (with assisted cooling modifications 3A)
- Others string variable, co-processor (option)



Program development environment

Hardware

- Personal computer — OS: Microsoft Windows supported
- Programming cable — Connect to a MPC-684 and a personal computer by RS-232.
  If the personal computer doesn't have a RS-232 port, you should use a USB-RS converter

Software

- You can install software for MPC programming by "Setup Disk".
- You can down-load it from the ACCEL web site.

Main software

FTMW32E.EXE terminal software

FTMWE is the software which connects a MPC-684 and a personal computer
This is used for debugging your program it also reads/writes the program and point data for the PC.
This is indispensable to development.

SYSLDWE.EXE system loader

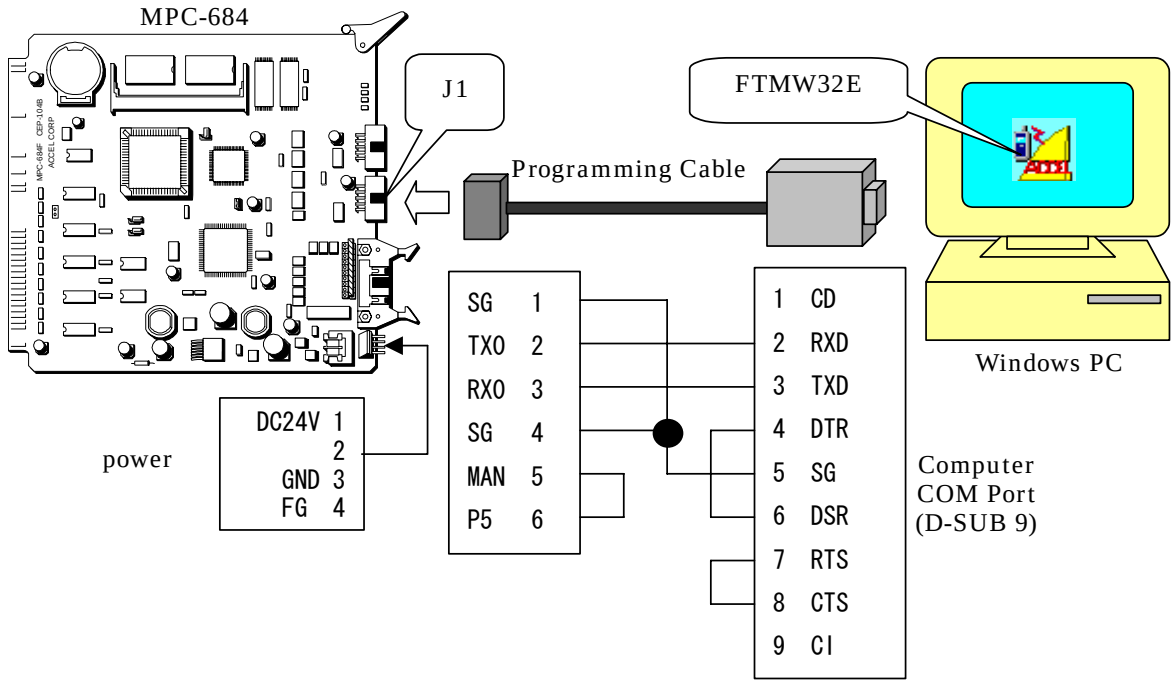
SYSLDWE is used for the MPC system version upgrade which rewrites flash-rom data to the MPC.

MPCEDE.EXE off line editor

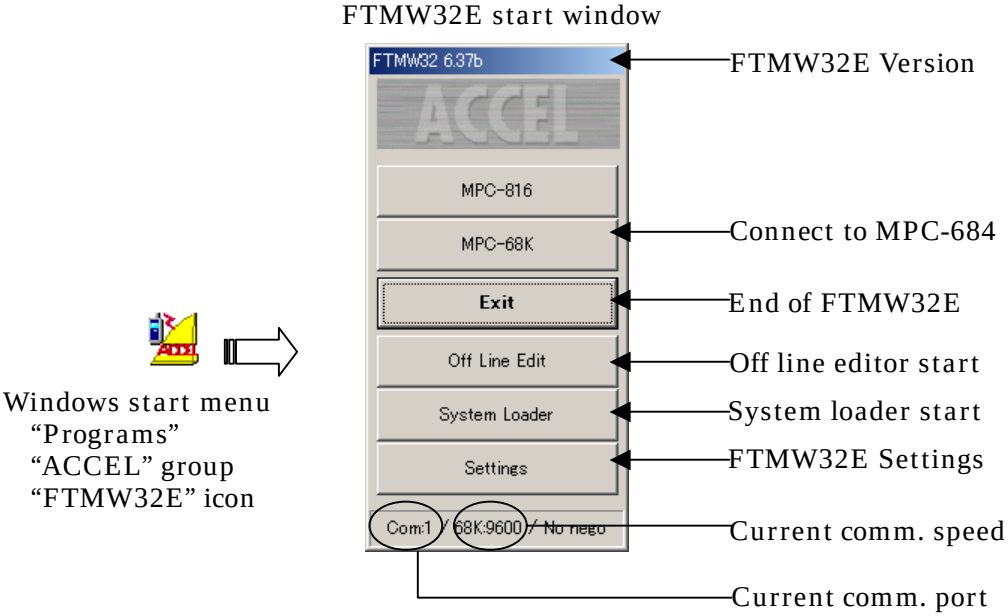
MPCEDE is an off-line editor only for the MPC.
It classifies control sentences, labels and comments by color.

How to connect

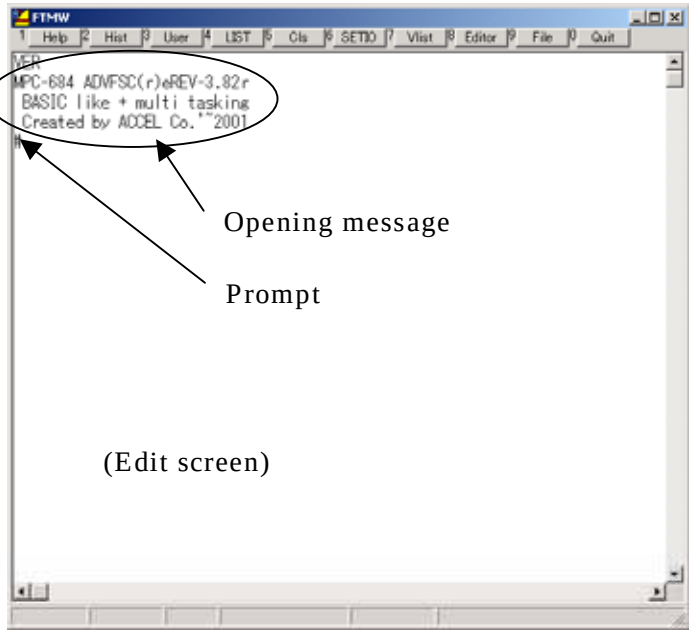
◆ Connect a personal computer to the MPC by the programming cable, then turn the MPC on.



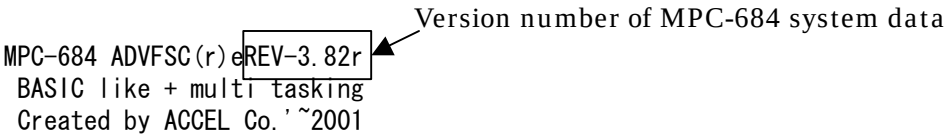
◆ After FTMW32E starts, set up the PC's comm port number and comm speed in the "Settings" window after that click "MPC-684" button.



◆ When it is able to connect normally, an opening message will display on the edit screen.



◆ The meaning of the opening message



Input commands

There are 3 types of input control.

- Direct executed commands: Single command statements executed immediately.
- Program executed commands: A series of commands executed from a program after initiating the run command.
- Bilateral commands: Both directly executed commands and program executed commands. Most commands are bilateral.

can use both	direct executed	program executed
ON 0	LIST	GOTO
OFF 0	MPCINIT	GOSUB
PRINT A	ERASE	IF ~
MOVE	RUN	FOR ~ NEXT
etc	etc	etc

```
#ON 0<Enter>            /* direct execution     10 ON 0<Enter> is program (attached number)
#GOTO 100<Enter>        /* not happen anything
#10 MPCINIT<Enter>      /* don't write  "MPCINIT" command in the program.
```

I/O check

There are 2 ways to check input and output conditions.

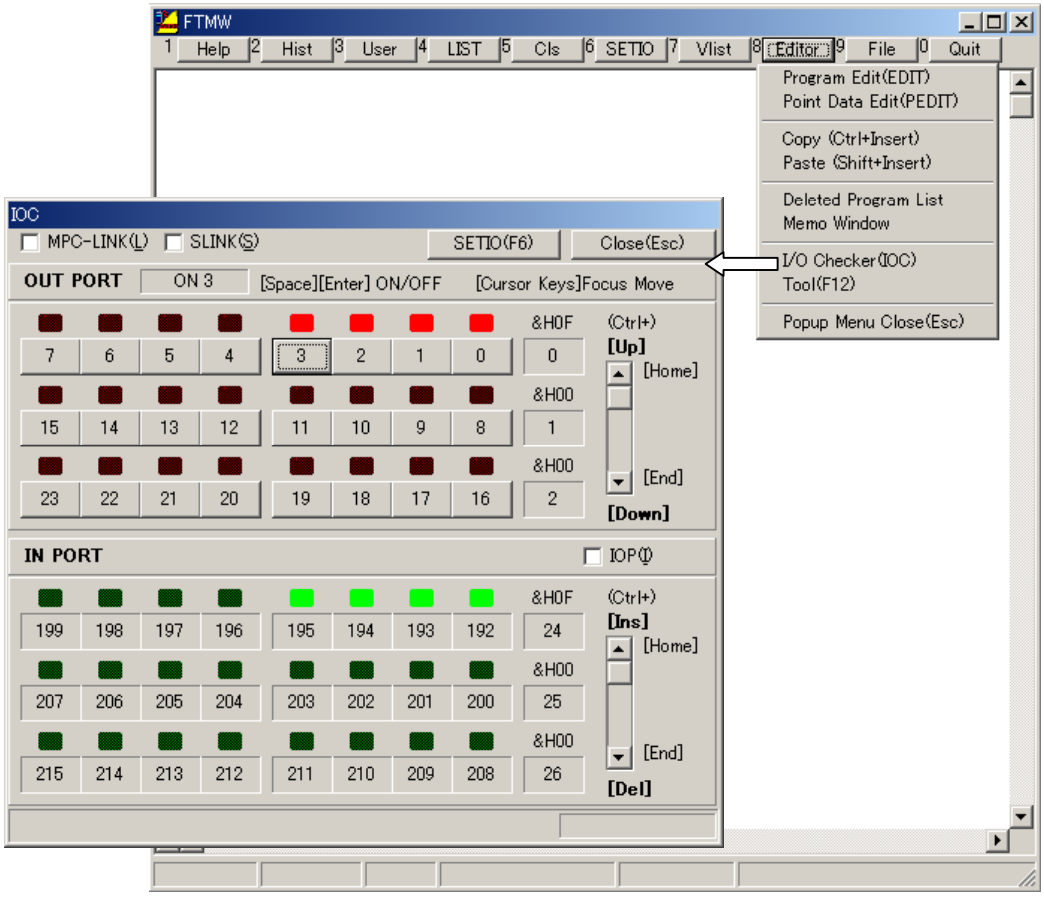
To check by command

- The following are commands used to complete a basic I/O check.

```
#ON 0                          /* turn the OUTPUT 0 on
#OFF 0                         /* turn the OUTPUT 0 off
#PR SW(0)                      /* The state of an INPUT 0 ("PR" is short for "PRINT")
 1                             /* 0=off,1=on
#PG &H400
#PRX HPT(0)                    /* check of MPG-314 origin sensor port (connecter J4)
 0005                          /* SX1(J4 No.11 pin) and SY1(J4 No.13 pin) are on
```

Check by the “I/O checker”

- To start the I/O checker push the F8 key or click on the ‘Editor’ button. It displays 3bytes at one time.



- You can check about MPG-314's INPUT by "INCHK_314" command.

```
#INCHK_314                    /* scan start
X_S1  ON   Y_S1  ON
Z_S1  __   U_S1  __
X_S2  __   Y_S2  __
Z_S2  __   U_S2  __
XIN2  __   YIN2  __
ZIN2  __   UIN2  __
XIN3  __   YIN3  __
ZIN3  __   UIN3  __
X-INP __   Y-INP __
Z-INP __   U-INP __
X-ALM __   Y-ALM __
Z-ALM __   U-ALM __
XLMT+ __   XLMT- __
YLMT+ __   YLMT- __
ZLMT+ __   ZLMT- __
ULMT+ __   ULMT- __
#                             /* scan stop by any key
```

How to input program

◆ To write a program include a line number before the command.

Press the enter key after writing each command, so it can be transmitted to the MPC.

If error messages come out, please check whether there is any mistake in grammar and input again.

```
10 'this is sample program<Enter>              /* comment
20 DO<Enter>                                   /* control statement
30 FOR I=0 TO 48<Enter>                        /* repeat
40 ON I : TIME 100<Enter>                      /* multi statement
50 NEXT I<Enter>
60 LOOP<Enter>
70 pri I<Enter>
.....This command is not supported. m(__)m ──┐ if error then correct and input again
70 PRINT I<Enter>                            <┘
```

Multi statement

- The MPC can differentiate between statements if they are separated by a colon (:)

```
1000      WAIT SW(0)==0 : ON 0 : TIME 100 : OFF 0 : TIME 500
```

Comment

- You can write comments after the statements using single quotation marks.
- If you write comments after the commands, the statement is automatically changed into a multi-statement.

```
#40 ON 0 'COMMENT<Enter>                          ┐
LIST 40 1<Enter>                                  │
40        ON 0 :                        'COMMENT <┘ multi-statement
```

- <Caution> If you write a colon in a comment statement the MPC will execute it like other commands.

```
10      'COMMENT  : ON 0       /* "ON 0" will execute
```

Label

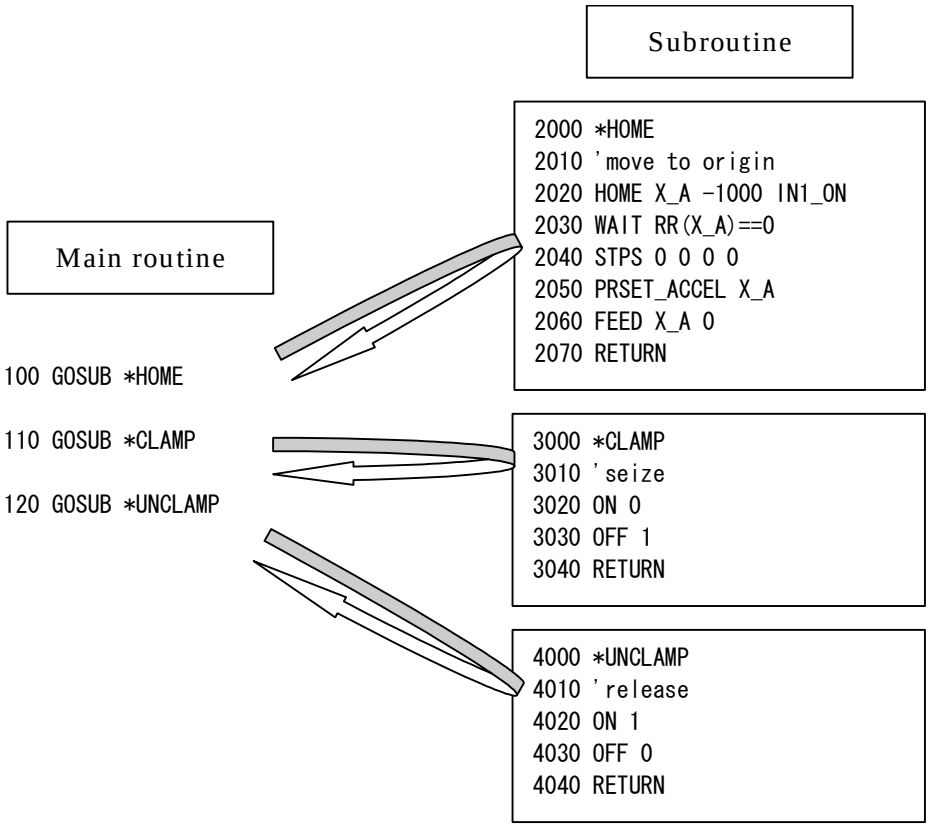
- The statement which has an asterisk (*) attached to the head is a label. Don't use a space in it.

```
10      *MAIN                                     /* label
20        IF SW(0)==0 THEN : GOTO *MAIN : END_IF
```

Subroutine

- A subroutine is a piece of code, much like any code in a program. A subroutine has an entry point and an exit point. At the entry of a subroutine, the system remembers where to resume execution when the subroutine finishes. When the subroutine finishes, the system resumes execution at this location.

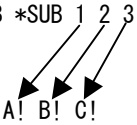
- If you make subroutines for each job, you can access them from the main routine, it will be easier to read and customize a program.



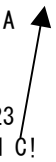
Parameter of subroutine, return value

- You can set a subroutine with parameters. The parameter values are activated by the main routine using the 'VAR' command. The command 'RETURN' ends the subroutine's program and commences the main routines next step.
- If the value combines with a local variable, some tasks can share the subroutine.

```
10  GOSUB *SUB 1 2 3     /* send parameter values
20  END
30  *SUB                 /* A!,B!,C! (with  '!' ) are local variables.
40  _VAR A! B! C!        /* get parameter values in the subroutine.
50  PRINT A! B! C!
60  RETURN
```



```
10 GOSUB *SUB
20 _RET_VAL A          /* get return value from the subroutine
30 PRINT A
40 END
50 *SUB
60  C!=123
70 RETURN C!           /* C! = return value
```



Edit of a program

◆ Explanation of useful operations for editing a program

LIST display

- 'LIST' is the command used most.

```
LIST [n,m]
  n: Start line number or LABEL
  m: The number of lines to display
```

- You can view the program's first block of code by entering 'LIST' (figure 16-1), if you enter 'LIST' once more it will display the next block. And so on consecutively.
- You can specify a line number or label to view by inputting it in the first parameter. (figure 16-2,16-3)
- You can specify the number of lines displayed in the 2nd parameter. (figure 16-4)
- You can view the program from the beginning by entering 'LIST 0'. (figure 16-5)

[16-1]
```
#LIST
10        IF SW(192)==0 THEN
20          GOTO *MANU
30          ELSE
```

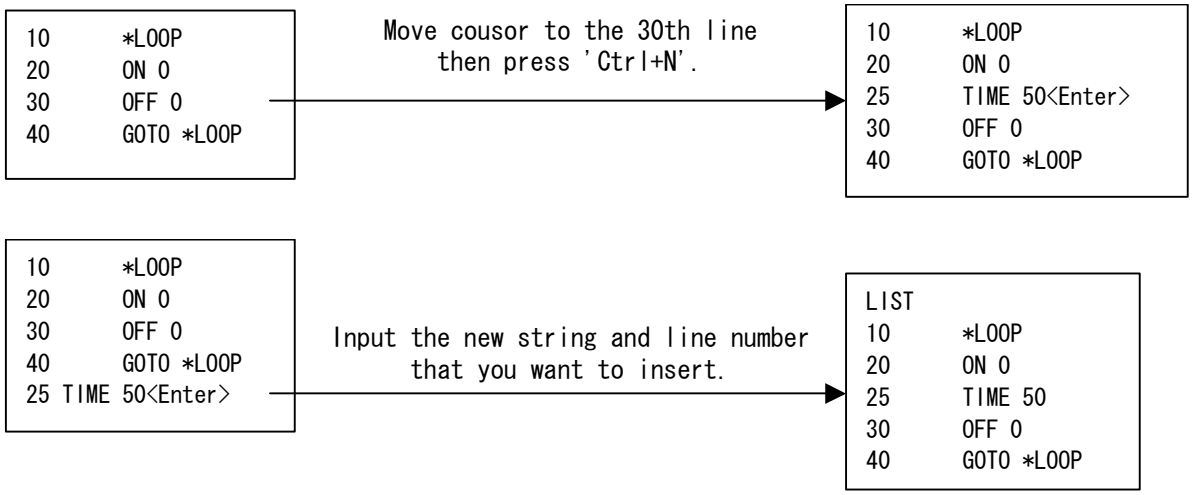
[16-2]
```
#LIST 40
40        GOTO *AUTO
50        END_IF
100     *MANU
```

[16-3]
```
#LIST *MANU
100     *MANU
110     'MANUAL JOB
200     *AUTO
```

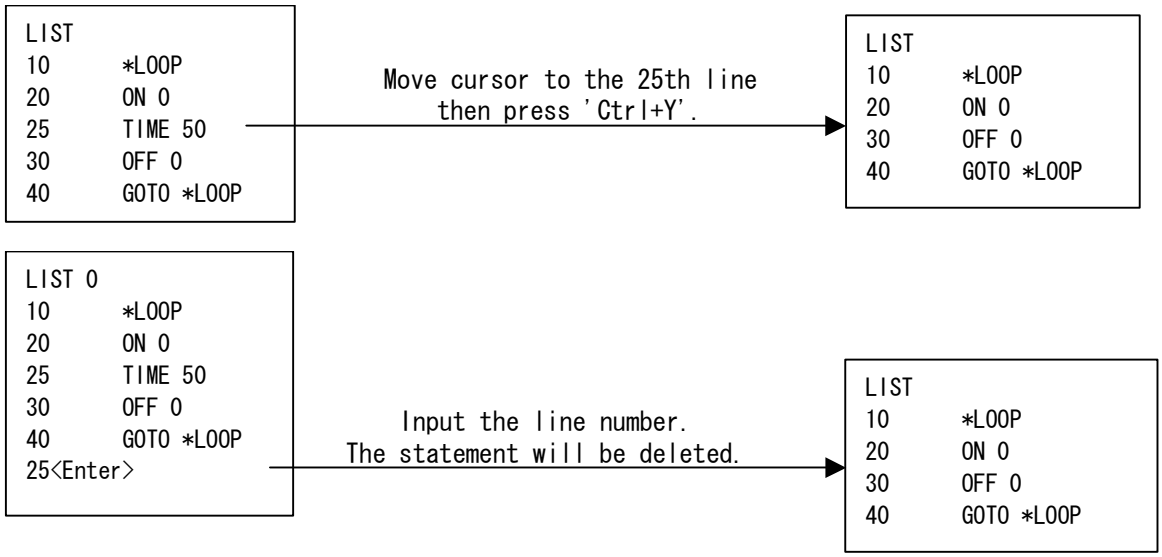
[16-4]
```
#LIST *MANU 2
100     *MANU
110     'MANUAL JOB
```

[16-5]
```
#LIST 0 20
10        IF SW(192)==0 THEN
20          GOTO *MANU
30          ELSE
40          GOTO *AUTO
50        END_IF
100     *MANU
110     'MANUAL JOB
200     *AUTO
210     'AUTO JOB
```

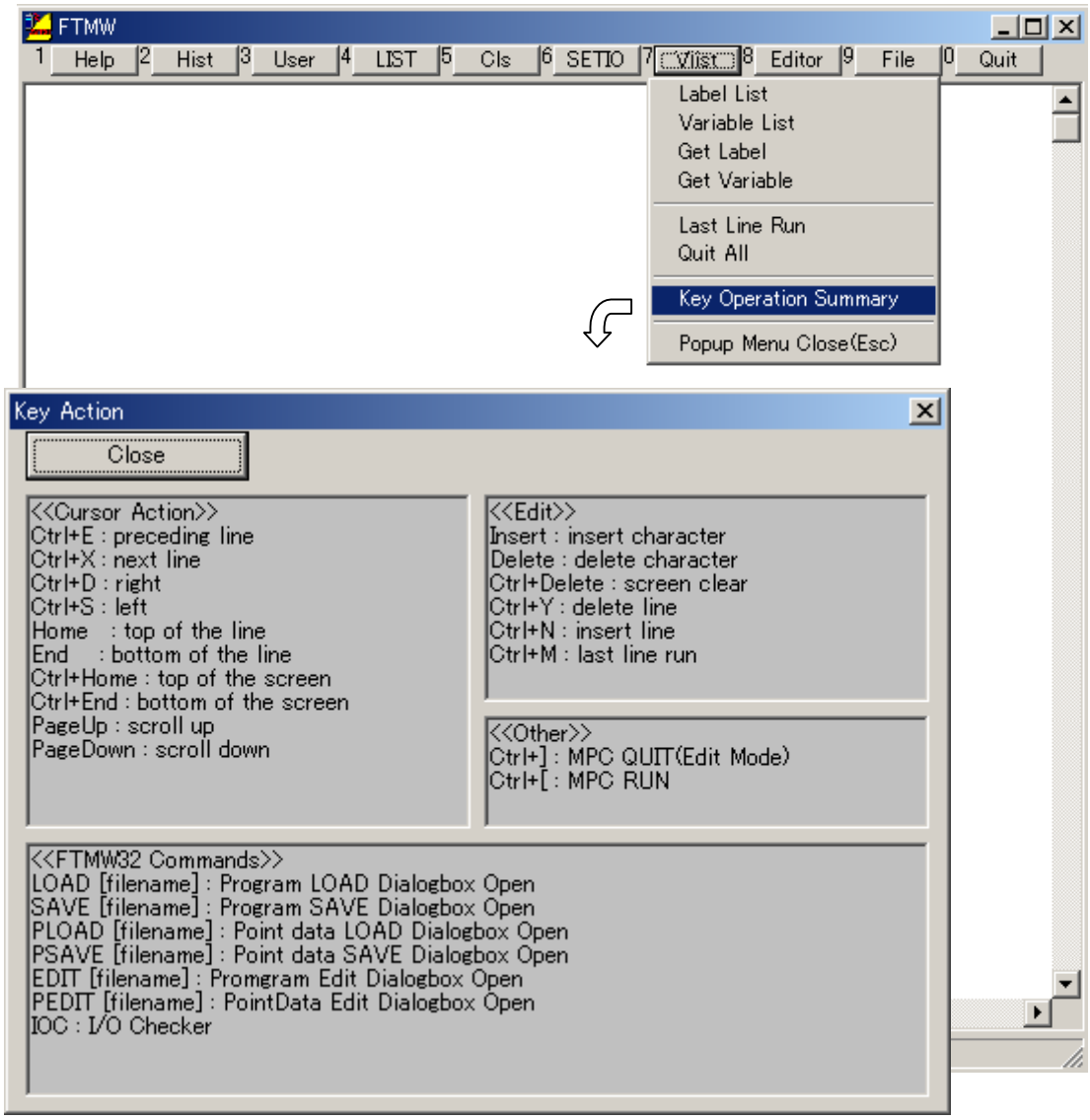
Insert statement



Delete statement



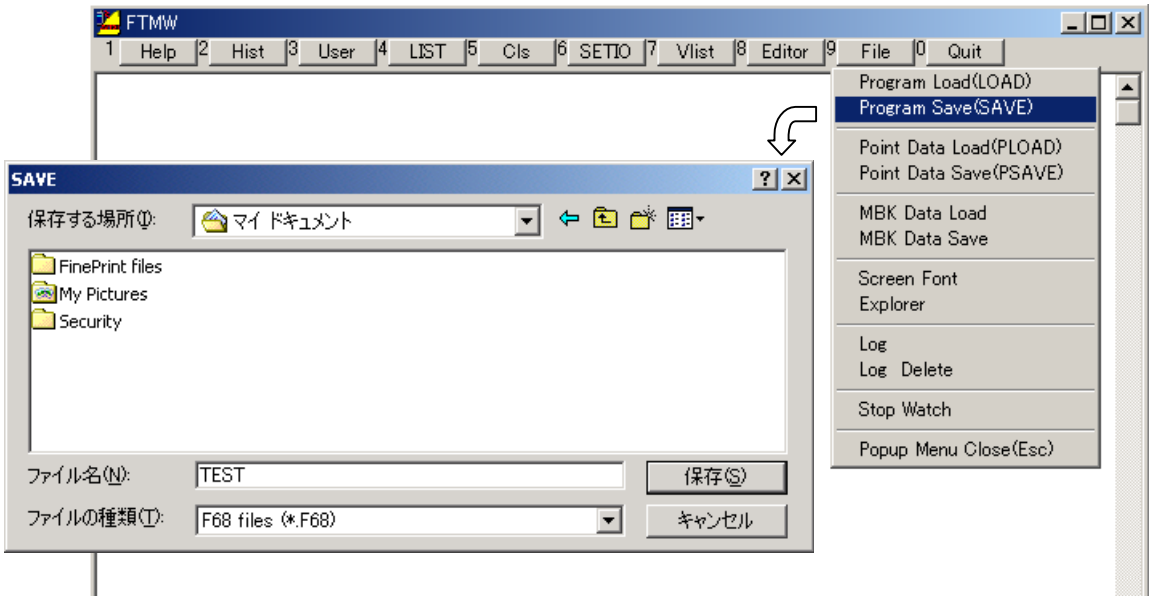
Other key operations



How to program save to the PC, load from the PC

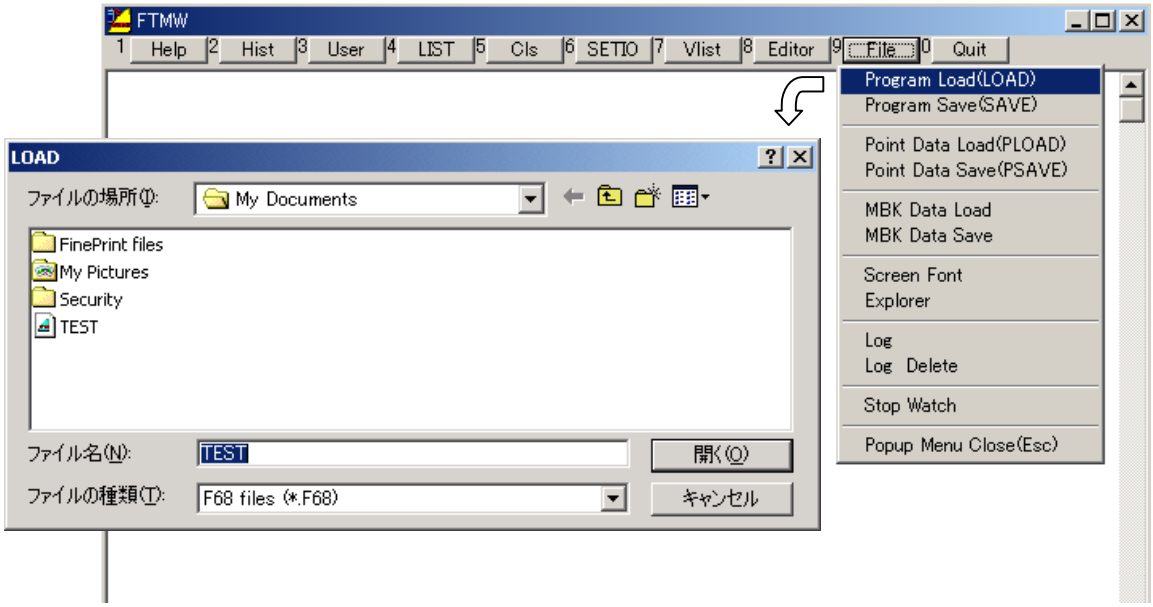
Program save to the PC

- You can save the program to your PC by pushing 'F9' then selecting 'Program Save' from the menu.
- The program is saved under the extension 'F68'.
- Saved programs don't have line numbers.



Program load from the PC

- You can load the program from your PC by pushing 'F9' then selecting 'Program Load' from the menu.
- Line numbers are in intervals of ten by default. If it exceeds 6000 it will be 'RENUM' to intervals of 5 automatically.



Off line editing

- Unless FTMW is connected with the MPC, it cannot be used, however you can create a program off-line using 'MPCED' or another common editor.

Printing

- FTMW cannot print a program. Please print the file saved in the PC by MPCED or another editor.

I/O control

- You can control an I/O Port for every bit or byte.
- Bit operating is for simple device control. ex. solenoid valves, relays and switches etc.
- Byte operating is used for reading DSW, data exchange with external devices etc.

Bit control

```
ON 0                                        /* output #0 turn on
OFF 1                                       /* output #1 turn off
WAIT SW(192)==1                             /* wait for input #192 to turn on
IF SW(1)==1 THEN : GOTO *LABEL : END_IF     /* conditional branch
ON -1                                       /* negative number = memory I/O.
WAIT SW(-2)==0                              /*
```

Byte control

```
OUT 170 0                                   /* 170(decimal) is outputted to bank #0.
A=IN(0)                                     /* the value of bank#0 loads to variable A.
OUT A B                                     /* the value of A loads to out-port B.
WAIT IN(1)==255                             /* wait to reach 255.
IF IN(2)==&H0F THEN : GOTO *LABEL : END_IF      /* conditional branch
OUT &HFF -1                                 /* negative number is memory I/O.
WAIT IN(-1)==&HAA                           /*
```

Variable, Array variable, String variable, Memory I/O

- MPC-684 has a lot of variables, array variables, string variables and memory I/Os. These are in integers of 4 bytes.
- Variables and array variables are used for calculation, counting as usual. Memory I/Os are usually used for interlocking tasks.
- Most commands and functions can use constants, variables and array variables in their parameters.
- You can control memory I/Os as usual by using ON, OFF, OUT, SW commands.
- You can view their current state by entering 'VLIST'.

Variable

- You can set the parameters of a command using variables or numbers.
- Parameters, commands, functions, constants and reservation strings cannot share the same name.
- You don't need to define them in the program as 'Language C', but you have to initialize variables in the program.

```
10          SOL1=0              /* SOL1 is a variable
20        ON SOL1               /* turn on
```

- You can make constants by 'CONST'.

```
10        CONST SOL1 0
20        ON SOL1
```

- You can't change constant's value.

```
10        CONST SOL1 0
20        ON SOL1
30          SOL1=1              /* changes the value.
#RUN
#30     .....Attempted modification of a constant.
```

Local variable

- The local variables are available for each task. They are assigned to the specific memory areas of each of the individual tasks. Each of the tasks can share a subroutine.

```
A!=B!+C!                        /*  '!' indicate local variable.
```

Array variable

- A DIM command secures specific array variable partitions in the memory.

```
10        DIM ARRY(100)        /* secure ARRY(0)~ARRY(99)
20        FOR I=0 TO 99
30            ARRY(I)=I
40        NEXT I
```

- You can use two-dimensional arrays.

```
10        DIM ARRY(2,3)
```

- Point data is also a kind of array variable.
  You can save (load) the point data to the PC. The PLS command displays the point data list.

```
#NEWP                          /* point data clear
#X(1)=999
#Y(1)=998
#U(1)=997
#Z(1)=996
#PLS
p1      X= 999 Y= 998 U= 997 Z= 996
  ---
p21     X= 0 Y= 0 U= 0 Z= 0
p22                            /*  'Q' key quits display. other key continue
```

String array variable

- There is a string array variable named 'AR$'.
- A DIM_AR$ command secures memory for AR$.

```
10        DIM_AR$ 35 4000      /* AR$(0)~AR$(3999): 4000 variables
20          AR$(3000)="ACCEL"
730        PRINT AR$(3000)
#run
ACCEL
```

- Caution: A string array variable shares the point data memory area.

```
#DIM_AR$                              /* just 'DIM_AR$' = confirm using memory
Length=35 Count =4000 P(MAX)=4249     /* 35 characters * 4000 arrays, 4249 available point data
```

Memory I/O

- A negative number denotes memory I/O.

```
ON - 1                         /* turn on bit -1
WAIT SW(-1)==1                 /* wait for bit to turn on
OUT 255 - 2                    /* 255 loads to bank -2 (bit - 9~-16)
```

Calculation

+	addition	A=B+C
-	substruction	A=B-C
*	multiplication	A=B*C
/	division	A=B/C
%	surplus	A=B%C
&	logical multiplication (AND)	A=B&C
|	logical addition (OR)	A=B|C
¥	exclusive-OR (XOR)	A=B¥C

◆ The integers are 4 bytes in value. The final value is always truncated to it's decimal point.

```
10          A=5
20          B=A/2
30        PRINT B
RUN
2
```

◆ The calculation below is in accordance with general arithmetic operation.

```
10          A=2 :   B=3 :   C=4
20          D=A*(B+C)
30        PRINT D
RUN
14
```

◆ You can carry out floating-point operations and trigonometrical function operations using a coprocessor (option).

Pulse generation

- This chapter introduces pulse generating using the MPG-314 board.
- It explains how to utilize the MPG-314 for an XY robot.

Initial settings

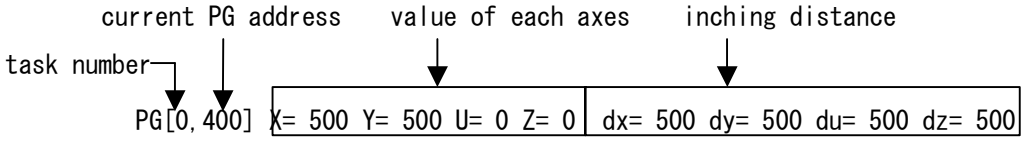
- Initialize the setting on the MPG-314 board before using it.
- At first you have to assign the MPG-314 board to any Tasks by using the PG command. Next, you set the MPG-314 board using ACCEL, FEED etc.
- You can do this by direct command, but you have to write the commands in your program.

For example

```
PG &H400                          /* MPG-314(address is &H400,it sets by DSW)select.
ACCEL ALL_A 5000                  /* maximum speed set
FEED ALL_A 0                      /* drive speed set
INSET_314 ALL_A ALM_ON|INP_OFF    /* In port set. 'ALARM' enabled on signal 'ON', 'INPOSITION'
                                     enabled on signal 'OFF'.
STPS 0 0 0 0                      /* setting the present position
```

How to check operation by TEACHING MODE.

- You can easily check about pulse generating by using the TEACHING MODE. FTMW is set to TEACH MODE with the T key.



- 'Inching' distance can be changed with the 0~3 key. (Their values are set by SET command.)

  [default] key0=10puls / key1=50puls / key2:100puls / key3:500puls
- Each axis can be moved by using the X,x,Y,y,U,u,Z,z keys.
- If you hit the P key, FTMW requires you to input a point number.
- You can change the PG number by using the Tab, +, - keys.
- Exit with the Q key from TEACH MODE.

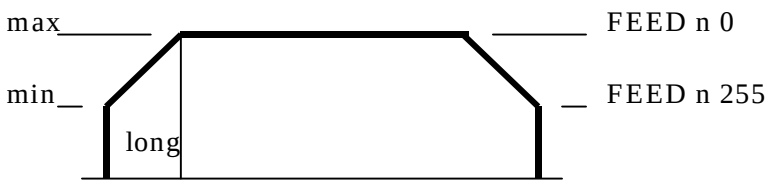
Setup of maximum high speed and acceleration.

- Maximum high speed and acceleration can be set by using the ACCEL command. Drive speed can be decided by using the FEED command.
- Format

```
ACCEL [n] [s] max [long min]
  n: axis constant X_A~Z_A
  s: S-curve acceleration parameter - 1~-100
  max: maximum high speed 1~4Mpps
  long: acceleration / deceleration pulse count
  min: start up pulse rate

FEED n m
  n: axis constant X_A~Z_A
  m: drive speed 0~255
```

- The relationship between 'ACCEL [n] [s] max [long min]' and 'FEED n m'

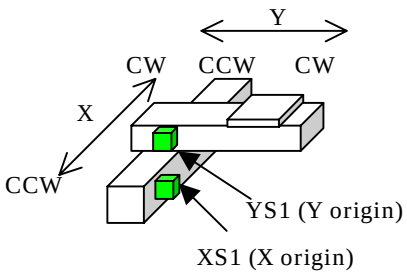


Move to the origin (HOME)

- HOME is the command for programming the movement of a robot back to the staring point of its action.

- Format

```
HOME n rate cond [n1 rate1 cond]
  n: axis constant X_A~Z_A
  rate: pulse rate (pps)
  cond: input condition INx_ON~INx_OFF
```



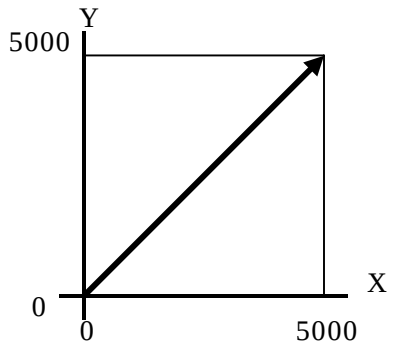
XS1=the 11th pin of J4 connector on the MPG-314 board.
YS1=the 13th pin of J4 connector on the MPG-314 board.

```
10    PG &H400
20    ACCEL ALL_A 5000
30    FEED ALL_A 0
40    GOSUB *HOME
50    END
60  *HOME
70    RMVL 500 500 0 0              /* move in an opposite direction from origin (CW)
80    WAIT RR(ALL_A)==0             /* wait until moving complete
90    HOME X_A -500 IN0_ON Y_A -500 IN0_ON       /* CCW movement until the sensor turns on
100   WAIT RR(ALL_A)==0             /* wait until movement complete
110   STPS 0 0 0 0                  /* setting current position to ZERO
120   PRSET_ACCEL ALL_A             /* ACCEL parameters restore
130   FEED ALL_A 0                  /* FEED reset
140   RETURN
```

Absolute coordinate movement

1) MOVL is the command for linear interpolation.

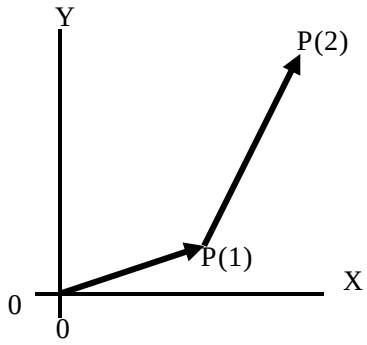
   You can give variables or constants to define the parameters.




```
10  PG &H400
20  ACCEL 5000       /* acceleration/deceleration setting
30  FEED ALL_A 128   /* speed setting
40  CLRPOS
50  MOVL 5000 5000 VOID VOID
60  WAIT RR(ALL_A)==0
```

2) Also, you can give a teaching point to the MOVE command parameter.

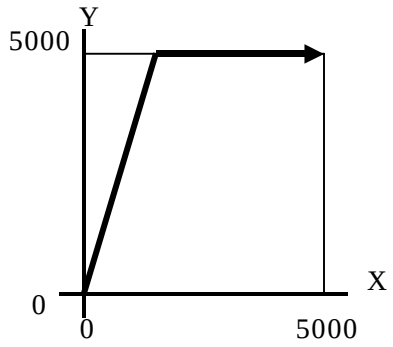
   The teaching points are set by using the 'teaching mode' or by programming.




```
10  PG &H400
20  ACCEL 5000
30  FEED ALL_A 128
40  CLRPOS
50  MOVL P(1)
60  WAIT RR(ALL_A)==0
70  MOVL P(2)
80  WAIT RR(ALL_A)==0
```

3) The MOVS command is similar to the MOVL command but it doesn't have linear interpolation.

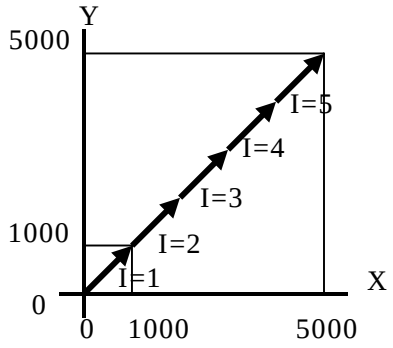
   You can set different acceleration (speed) on each axis.




```
10  PG &H400
20  ACCEL X_A 5000   /* X-axis acceleration/deceleration setting
30  ACCEL Y_A 10000  /* Y-axis acceleration/deceleration setting
40  FEED X_A 128     /* X-axis speed setting
50  FEED Y_A 0       /* Y-axis speed setting
60  CLRPOS
70  MOVS 5000 5000 VOID VOID
80  WAIT RR(ALL_A)==0
```

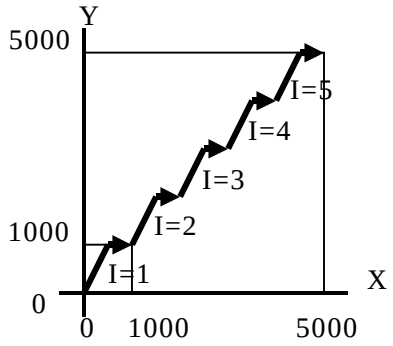
Relative coordinate movement

1) RMVL is the command for linear interpolation of relative coordinate movement.
   You can give variables or constants to define the parameters.




```
10  PG &H400
20  ACCEL 5000
30  FEED ALL_A 0
40  CLRPOS
50  FOR I=1 TO 5
60    RMVL 1000 1000 0 0
70    WAIT RR(ALL_A)==0
80  NEXT I
```

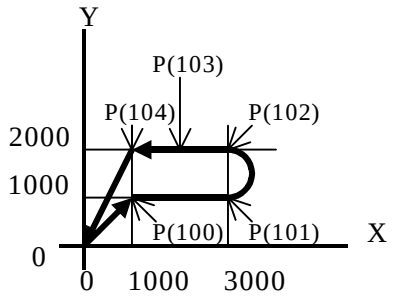
2) The RMVS command is similar to the RMVL command but it doesn't have linear interpolation.




```
10  PG &H400
20  ACCEL X_A 5000
30  ACCEL Y_A 10000
40  FEED X_A 128
50  FEED Y_A 0
60  CLRPOS
70  FOR I=1 TO 5
80    RMVS 1000 1000 0 0
90    WAIT RR(ALL_A)==0
10  NEXT I
```

Continuous interpolation

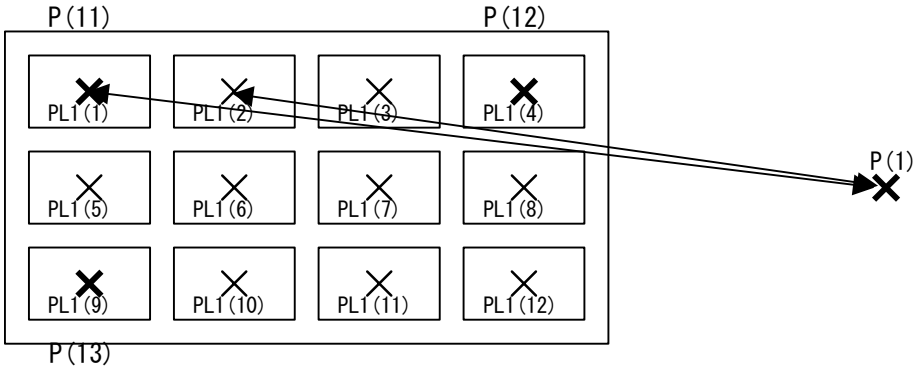
- The following figure is an example of continuous interpolation by using the MOVT command.
- It sets point data between the 10 line and the 50 line.
  In this case, P(100) is the base point of the coordinates. P(101)~P(104) are local points (relative coordinates to the base point).
- The MPC interpreter reads a MOVT command then executes it. The MPC interpreter reads the next program step while the previous step is being executed. If you use the I/O port to control an action in the middle of any program step (that is being executed) you have to consider the time gap between the working position and the next program execution step.
- The following program is output #0 turn on from P(101) to P(102).




```
10   SETP 100 0 0 0 0              /* the base point
20   SETP 101 2000 0 0 0
30   SETP 102 2000 1000 2000 500
40   SETP 103 1000 1000 0 0
50   SETP 104 0 1000 0 0
60   PG &H400
70   ACCEL 5000
80   STPS 0 0 0 0
90   MOVL 1000 1000 VOID VOID      /* move to the base point
100  WAIT RR(ALL_A)==0
110  HOUT X_A;DS_DACL              /* acceleration/deceleration disable
120  MOVT X_A|Y_A P(101)
130  MOVT X_A|Y_A P(102) CCW
140  MOVT X_A|Y_A P(103)           /* dummy point for output #0 turn off at P(102)
150  ON 0                          /* output #0 turn on from P(101) to P(102)
160  MOVT X_A|Y_A P(104)
170  OFF 0
180  HOUT X_A;EN_DACL              /* acceleration/deceleration enable
190  WAIT RR(ALL_A)==0
200  MOVL 0 0 VOID VOID
```

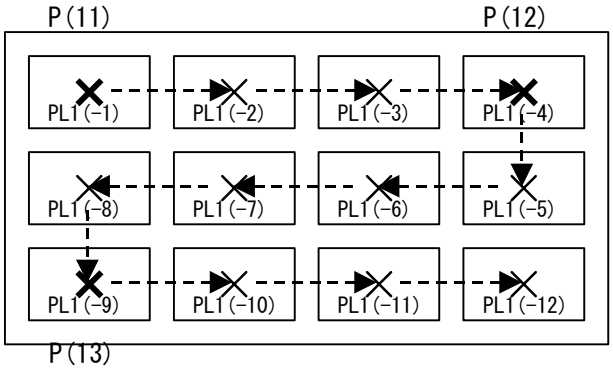
PALET declaration

- PALET is the command for moving between palettes.
- The PALET command calculate all working points (PLn(N)) on a palette by using 3 teaching points.




```
10  PG &H400
20  ACCEL ALL_A 5000
30  PALET1 P(11) P(12) P(13) 4 3    /* declare PALET
35  MOVL 0 0 VOID VOID
37  WAIT RR(ALL_A)==0
40  FOR N=1 TO 12
50    MOVL P(1)
60    WAIT RR(ALL_A)==0
70    MOVL PL1(N)                   /* move to point (N) on the PALET
80    WAIT RR(ALL_A)==0
90  NEXT N
```

- When the 'N' number is a negative value, the motion becomes as shown in the figure below. The movement between columns becomes smaller, and thus the speed increases.



Conditional stop

Pulse generation stops after sensors receive data indicating the need for the stop condition to take effect.

- 1) The STOP command executes after the MOVL command. You can use normal input ports (on the MIP, IOP board) for the stop parameters.

```
10  PG &H400
20  ACCEL 1000
30  FEED ALL_A 0
40  MOVL 5000 5000 VOID VOID
50  WAIT SW(192)==1
60  STOP ALL_A STP_I          /* STP_I : without deceleration, STP_D : with deceleration
70  WAIT RR(ALL_A)==0
```

- 2) You can set the stop conditions before executing the MOVL command. In this case you have to define specific input ports (IN3, LMT, ALM) on the MPG-314 board for the stop parameters.

```
10  PG &H400
20  ACCEL 1000
30  FEED ALL_A 0
40  STOP ALL_A IN3_ON     /* set stop condition. if X-IN3(J2 connecter pin1) turn on then stop.
50  MOVL 5000 5000 VOID VOID
60  WAIT RR(ALL_A)==0
70  IF RR(IN3_)<>0 THEN : PRINT "IN3stop" : END_IF         /* confirming reason for stop
80  STOP ALL_A VOID                   /* reset stop condition
```

Multi-task

- MPC-684 has 32 tasks (task0 : Main-task. task1~31 : sub-tasks)
- Main-task executes immediately after the RUN command (program mode) or MPC-684 power-on (auto mode). Sub-tasks are executed from main-task.
- There isn't any prioritization of sub task. A sub task can execute or cancel out other sub tasks.

The commands for multi-task

- FORK,END,QUIT

The FORK command executes a sub-task.

The task is cancelled by the END command included in that task or by using the QUIT command in a different task.

- PAUSE,CONT

The PAUSE command pauses other tasks execution.

The CONT command enables the task to continue after it has been paused by the PUASE command stop.

```
10   FORK 1 *TASK1              /* sub-task1 execute
20   WAIT SW(192)==1
30   QUIT 1                     /* sub-task1 destroy
40   OFF 0
50   END
60 *TASK1
70   DO
80     ON 0 : TIME 500
90     OFF 0 : TIME 500
100  LOOP
```

RS-232 communication

- MPC-684 has two RS-232 communication ports for interfacing to external devices. You can control the ports by using commands.
- One of the ports supports interfacing with the GP touch-panel (Digital corporation).
- You can add more communication ports by using the MRS-402 board (max 4ch.)

Command for communication

- CNFG#n channel n initialize
- PRINT#n channel n transmit
- INPUT#n channel n receive

```
CNFG#0 "9600b8pns1XON"      /* 9600bps 8bit parity-none 1stop XON/OFF
INPUT#0 a$                  /* string receive
  b$="&H"+a$                /* string processing
  v=VAL(b$)                 /* conversion string to variable
PRINT "&H"+a$ "=" v         /* display to monitor screen ( the FTMW )
PRINT#0 HEX$(v)+"¥n"        /* transmit
```

Debugging

Basic debugging (run/stop/check)

- One useful basic process of debugging of the MPC is by executing the program, stopping it and then checking previous tasks program steps.

  For instance, if a machine that is in operation stops, you can look for the cause of stoppage.

  FTMW displays the stop lines of each task by using 'Ctrl+M'.

```
10        ON 0
20        WAIT SW(192)==1
30        OFF 0
#RUN                          /* RUN -> if the machine stopped -> program stop by using 'Ctrl+A'
        *0   [20]
#                             /* Display stop line(s) by using 'Ctrl+M'
TASK0 20     WAIT SW(192)==1  /* Why did program stop here?
```

Use the PRINT command

- You can monitor the variables, I/O, etc. by using the PRINT command in the program.

```
10          C=0
20        DO
30            C=C+1
40          PRINT "count=" C   /* monitoring variable C
50        LOOP WHILE C<3
#RUN
count=1
count=2
count=3
```

Subroutine execute

- You can execute any subroutine by using the RUN command.

```
10        DO
20          GOSUB *FLICK
30        LOOP
40      *FLICK
50        ON 0
60        TIME 50
70        OFF 0
80        TIME 50
90        RETURN
#RUN *FLICK                                        /* execute the subroutine *FLICK
#90     .....Return not called from a subroutine   /* end of the subroutine
```

How to know the cause of machine stoppage

- When the machine stops while in automatic operation, please connect the MPC-684 to FTMW without turning off the power supply of the machine.

```
<< The machine stopped! Why? Connect to FTMW soon! >>
VER
MPC-684 ADVFSC(r)eREV-3.82v
 BASIC like + multi tasking
 Created by ACCEL Co.'~2001              /* MPC-684 connecting to FTMW
#MON                                     /* You can know condition by using the MON command
        *0   [20]
#                                        /* FTMW displays condition by using 'Ctrl+M'
TASK0 20        WAIT SW(192)==1          /* task0 stopped at 20, waiting switch 192 turns on
#LIST 0
10        ON 0
20        WAIT SW(192)==1
30        OFF 0
```

How to read the programming port output log.

- The FTMW displays RS-232(ch1:program port) log data when you use the LOG command. You can find runtime-errors by using it.
- The MPC-684 has 1k byte memory for logging that ring-buffer.
- You can buffer clear input 'LOG 0'.
- The MPC-684 can't log runtime-errors that happened during a task0.

```
LIST
10        FOR I=1 TO 2
20          PRINT I
30        NEXT I
#LOG 0                          /* clear log data
#RUN
1                               /* PRINT output under executing
2
#LOG                            /* watch log data

1                               /* log data in the memory
2

#
```

Special program

- The prompt returns to the FTMW when task 0 is ended using the END command. Then you can execute the commands for an I/O and variable's value, checks.

- <Caution>

  *The program currently being executed stops when you modify the program.

  *You can't use Ctrl+A for program stop. Use Ctrl+] instead.

```
10        FORK 1 *JOB1
20        END                 /* the TASK0 is END
30      *JOB1
35        DO
40          ON 0
50          TIME 50
60          OFF 0
70          TIME 50
80        LOOP
#RUN                          /* program execute
#                             /* The prompt returns. TASK1 is running
TASK0 20          END         /* Crtl+M display current running steps
TASK1 70          TIME 50
#PRINT SW(0)                  /* You can use the PRINT command for I/O check
1
#       *0   [20]             /* program stops by Ctrl+]
        *1   [50]
#
```

Use touch panel

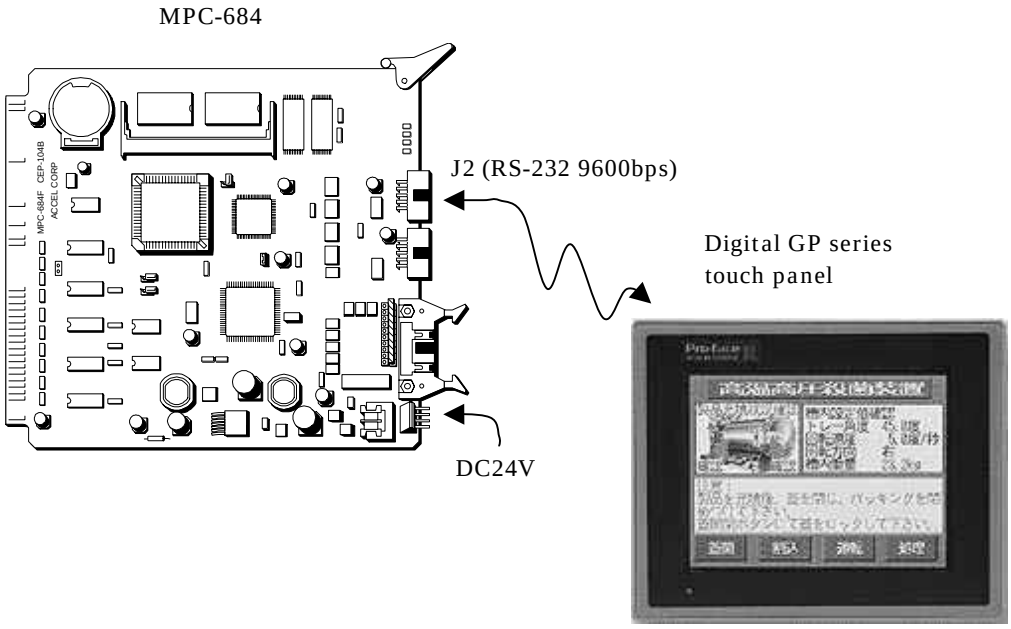
- The MPC-684 supports the 'Direct access protocol' on the Digital Electron Corporation's GP series touch panel.
- You can select MBK-RS (by using J2 on the MPC-684) or the MBK-SH board for communication to the GP.
- The MPC-684 has a 1000 words data area and a 100 words I/O area of memory capacity.

When you use MBK-RS

- You have to write 'PROTOCOL MEWNET' at top of the program.

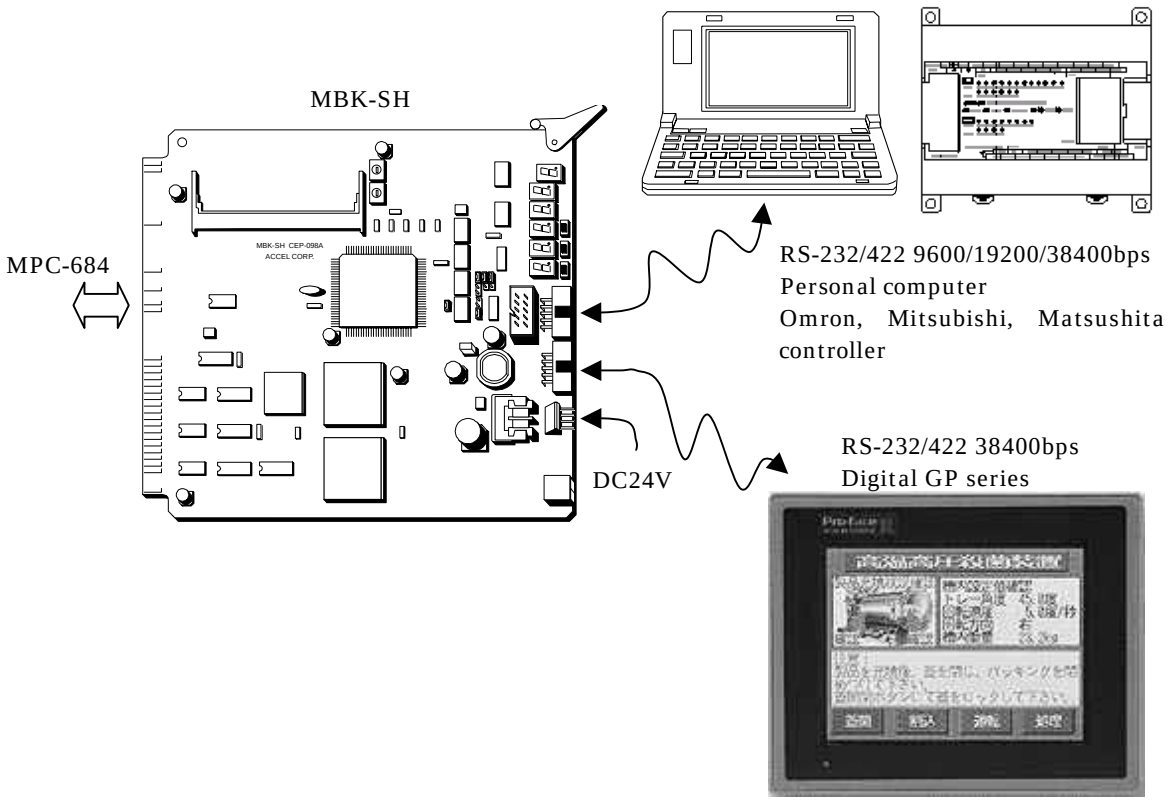
```
10  PROTOCOL MEWNET
```

- You can't use the MBK-RS and the RS-232 CH0 at the same time.



When you use MBK-SH

- The MBK-SH connects to the touch panel at RS-422 38400bps.
- The MBK-SH board is an independent CPU board, therefore doesn't put strain on the main CPU board.
- You can use the MBK-SH for communication with another controller.



Command List

I/O

?	Reading of Bit (HSW Alternative)
ALT	Switch ON/OFF
CLR_OUTP	Initialize Output (specified the board)
HIN	Parallel Input
HIN	Parallel Input (Read 4 bytes)
HOUT	Pulse Board General Output
HPT	Pulse Board General Input
HSW	Read Bit
IN	Parallel Input
IO	I/O Monitor
IOR	Read BUS
IOW	Write BUS
OFF	Output off
ON	Output on
OUT	Parallel Output
P_HSW	Output/Input of MPC-684 J4 Connector
P_IN	Output/Input of MPC-684 J4 Connector
P_IN	High Speed Input MPC-684 J4Connector
P_OFF	Output/Input of MPC-684 J4Connector
P_ON	Output/Input of MPC-684 J4Connector
P_OUT	Output/Input of MPC-684 J4Connector
P_SW	Output/Input of MPC-684 J4Connector
SENSE_SW	Output Operation on Input Detection
SETIO	Initialize Output
SW	Bit Input
WS0	Input with Time Out
WS1	Input with Time Out

MBK-SH/RS

ADD_MBK	Data Increment
CHR$	Conversion from ASCII to String
DIMCPY	Copy Arrays
LD_M	Memory Bulk Copy (MPC to MBK)
MBK	Reading of Data Area
MBK	Verification of Backup Status
S_MBK	Write Data Area
S_MBK	String Forward
SV_M	Memory Bulk Copy (MBK to MPC)

MPG-314 Exclusive

ACCEL	Creation of Acceleration/Deceleration Table
ERR_PAUSE	Control of Task on Error
FEED	Speed setting
FEED	Speed setting(Minute Setting)
HOME	Stated Low Speed Return to the origin
HOUT	Port Output
HOUT	Control Register

HPT	Read Input Port
INCHK_314	Input Monitor
INSET_314	Set Input Port Function
M_RMVS	Asymmetric Acceleration/Deceleration Move
MOVL	Linear Interpolation (Absolute Coordinate Move)
MOVS	Axis Independent Pulse Generation (Absolute Coordinate Move)
MOVT	Continuous Interpolation (Absolute Coordinate Move)
PG	PG Declaration
PLSC	Pulse Generation at a Constant Speed
PRSET_ACCEL	Reset ACCEL Parameter
RANGE	Operative Restriction
RMVC	Infinite Pulse Generation
RMVL	Linear Interpolation (Relative Coordinate Move)
RMVS	Axis Independent Pulse Generation (Relative Coordinate Move)
RMVT	Continuous Interpolation (Relative Coordinate Move)
RR	Read Register
STOP	Conditional Stop
STPS	Specify Current Position
STPS	Set Counter
U	Read Counter
WARP	Warp Jump
X	Counter Reading
Y	Read Counter
Z	Read Counter

MPG-3202 Exclusive

CMND	X3202 Command Execution
REG	Read X3202 Register
REG3	Read X3202 Register
ST_REG	X3202 Register Writing

MPG-68K Compatible

ACCEL	Creation of Acceleration/Deceleration Table
BSY	PULSE Generation Status Input
CLRPOS	Clear current position
CURPOS	Display current position
FEDD	Speed setting
FEDH	Speed setting
FEDT	Speed setting
FEDZ	Speed setting
FEED	Speed setting
GO	4 axes simultaneous PULSE GENERATION
HOME	Return to the origin
HOMZ	Return to the origin
JMPZ	Gate motion move
JUMP	Gate motion move
LIMZ	Limit of Gate Motion
MOVE	XYU Absolute Coordinate Move
MOVZ	Z Absolute Coordinate Move
P	Point Data
PALET1	PALET Declaration
PALET2	PALET Declaration
PALET3	PALET Declaration

PALET4	PALET Declaration
PG	PG Declaration
PGSEL	PG Declaration
PL1	Palette Point
PL2	Palette Point
PL3	Palette Point
PL4	Palette Point
PULSE	Pulse Generation at a Constant Speed
Q_PAUSE	Quick Pause
RANGE	Operative Restriction
RM	4-axis Relative Coordinate Move
RMOV	XYU-axis Relative Coordinate Move
RMVZ	Z-axis Relative Coordinate Move
SET	Setting Inching Amount
SETP	Sets point data
SETPOS	Change Current Value
SHMZ	Setting Return to the Origin
SHOM	Setting Return to the Origin
STOP	Stop Pulse Generation
TEACH	Teaching Mode
U	U-axis Point Data
X	X-axis Point Data
Y	Y-axis Point Data
Z	Z-axis Point Data

RS-232

CNFG#0	Setting of Communication Mode
CNFG#2	Setting of Communication Mode
INP$#0	Read n Character
INP$#2	Read n Character
INPBLK#	Binary Fixed Format Input
INPUT	Data Input
INPUT#0	Data Input
INPUT#2	Data Input
INPUT$	Read n Character
LOF	Number of Character of Buffer
PRINT	Data Output
PRINT#0	Data Output
PRINT#2	Data Output
PROTOCOL	MBK-RS (Digital GP Direct Access)
PRX	Data HEX Display
PUT	Data Output
PUT#0	Data Output
PUT#2	Data Output
RS	Display Buffer
RSE	RS-232C ERROR
RSE	CH1 Character Input
SLOW	CH1 Character Transmission Interval

Calendar

date$	Get Date String
time$	Get Time String
timer	Get Time

Floating Point Operation

CALF	Arithmetics
GETF	Retrieve Data
PRF	Display Internal Data
SETF	Delivery of Data

System

FCLK	Change Clock Speed
MPC	MPC-816 COMPATIBLE

Timer

SYSCLK	System Clock
TIME	Wait Time
TMOUT	Set Input Time

Task Operation

CONT	Continue Task
FORK	Task Execution
LIFE_TIME	Set the Life Time of Task
PAUSE	Pause Task
QUIT	Stop task
RLS	Release Semaphore
RSV	Semaphore capture
SWAP	Abandonment of Execution
TASKN	Task Number Get

Debug

CNT	Continue Execution
DUMP	Display of Memory
FIND	Search String
FIX	Write to Flash ROM
LOG	Program Port Output Record
MON	Verification of Stop Status
RAM	Receipt of Return Value
ROM	Flash Rom Mode
RUN	Execute Program
TASK	Task Status Display
TOFF	Trace mode
TON	Trace mode

Bus Access

WIR	Word Reading
WOW	Word Writing

File Memory

P_LD	Read Point Data from Flash ROM
P_SV	Write Point Data in Flash ROM

Memory Access

PEEK	Readout of User Memory
POKE	Write User Memory

Maintenance

ERASE	Clear Program of Flash ROM
MEM	Memory Test
MPCINIT	Initialize RAM Area
TMON	Task Monitor
VER	Version Data Display

User Command

ADR	Obtain the address
COMSET	Setting of Command Name

Arithmetic

@	Logical Negation
ABS	Absolute value
AND	Logical Sum Expression (Logical Multiplication)
ATAN	Trigonometric Function
ATAN2	Trigonometric Function
CONST	Conversion from variable to constant
COS	Trigonometric Functions
DIM	Arrays Declaration
DIM	Arrays Declaration (two dimensional)
LET	Expression Execution
NOT	Complement
OR	Logical Merge (Logical Sum)
SFTL	Array Variables Rotate
SFTR	Array Variables Rotate
SIN	Trigonometric Functions
SQ	Square
SQR	Square Root
SWP	Switch Upper/Lower Bytes
TAN	Trigonometric Functions

Control Statement

BREAK	End of Control Flow (BREAK from IF Statement)
BREAK	End of Control Flow (BREAK from LOOP)
CASE	Conditional Branch Dependent On Value
CASE_ELSE	Conditional Branch Dependent On Value
DO	Loop
ELSE	Conditional Branch
END	End of Program
END_IF	Conditional Branch
END_SELECT	Conditional Branch Dependent On Value
FAST	Stop SWAP Function
FOR	Loop
GOSUB	Subroutine Call

GOSUB	Subroutine Call (Argument Pass)
GOTO	Unconditional Branch
IF	Conditional Branch
LOOP	Loop
NEXT	Loop
_RET_VAL	Receipt of Return Value
RETURN	Return
RETURN	Return Value Passing
SELECT_CASE	Conditional Branch Dependent On Value
THEN	Conditional Branch
UNTIL	Conditional Statement
_VAR	Receipt from Arguments
WAIT	Wait for a condition
WEND	Condition Loop
WHILE	Condition Loop

String

AR$	String Arrays
ASC	Conversion from String to ASCII
CHR$	Conversion from ASCII to String
DIM_AR$	String Arrays Declaration
DIMCPY	Copy Arrays
GET_VAL	Automatic Numeric Numbers Retrieve from String
HEX$	Conversion form String to Hex
LEN	Get Number of Character
STR$	Convert from Numeric Value to String
STRCPY	Copy String
VAL	Numbers from Numeric String
VER$	Version Data Get

Edit

DELETE	Delete of Program
FREE	Display Available Memory
LIST	Display Program
NEW	Initialize Program
NEWP	Initialization of point data
PLIST	Display Point Data
RENUM	Reset Sentence Number
TAIL	Maximum of Statement Number
VLIST	Program Reference Display

--- End Of Document ---